Quantitative model for choosing programming language for online instruction

Steven J. Sherman
Troy University

Ronald F. Shehane
Troy University

Dewey W. Todd
Troy University

ABSTRACT

Colleges are increasingly offering online courses, including computer programming courses for business school students. Programming languages that are most useful to students are those that are widely used in the job market. However, the most popular computer languages change at least every three years. Therefore, the language used for instruction in a business programming course must be re-evaluated regularly. This paper establishes a quantitative model for choosing the best language for an online business programming course. Factors considered in the quantitative model include platform compatibility, marketability of the language, object-oriented programming support, availability of an interactive development environment, and the availability of interactive online teaching tools and videos. The paper next presented an example of the quantitative model used for selecting a language for a new online business programming course at a selected university. The final section of the paper presents the results and conclusions of the analysis.

Keywords: Online Teaching, Programming Course, Programming Languages, Computer Language, Selecting Program Languages

INTRODUCTION

The foundation of a programming course is the language chosen for the instruction (Wu & Chang, 2011). Chen, et al. (2005) analyzed the historical trends of high-level programming languages. Their research showed that the top three programming languages changed at least every three years. This means that choosing the correct programming language for a business applications programming course needs to be re-examined regularly. Looking at it another way, a programming course that has been using the same programming language for more than three years may be out of date.

The selection of a programming language for introductory courses is a critical and often controversial process (Stroustrup, 2009). However, the development of quantitative models that guide the selection of programming languages for introductory computer programming courses are generally lacking in research literature, particularly as applied to online teaching environments. Clarke (2001) focused his research on the use of questionnaires to assess the cognitive aspects of languages in evaluating programming languages, but did not provide a complete quantitative model for evaluation. Gupta (2004) addressed the needs of beginning programming students, but did not provide a useful formal evaluation approach. Parker et al. (2006) developed a list of possible criteria for evaluation, but was missing details that would enable full use of the criteria for evaluating the acceptability of programming languages for course use. Articles by (Feldman, 1992, Gries, 1974, McIver 2002, Schneider, 1978) addressed various aspects of programming selection, but failed to provide a quantitative model that would guide the selection process. Farooq, et al. (2014) provide the most complete evaluation approach for language selection to date, but only address the technical and environmental aspects of languages and did not focus on teaching considerations nor online learner needs that should be considered in a more practical and useful evaluation tool for selecting a programming language to be used in an online environment.

How can faculty committees perform repeated evaluations of programming languages in a consistent way to ensure continuity and appropriate choices? This paper will address those questions by establishing a quantitative model for evaluating business programming languages for online instruction. The first part of this paper describes the challenges in teaching programming in an online environment. The second section of this paper presents a quantitative model for evaluating programming languages for suitability in an online learning environment. The third section demonstrates the application of this quantitative model in choosing a programming language.

ONLINE PROGRAMMING INSTRUCTION

In the 2000-01 school years, 89 percent of public two-year and four-year institutions were offering online courses. Of the four-year institutions, nearly half offered complete degree programs delivered exclusively online (Durrington, Berryhill, & Swafford, 2006). These statistics make it clear that distance learning is not a trend – it is the new reality and an increasing area of growth in education. (Allen & Seaman, 2010; Mosca, et al., 2010). While the growth of online education is clear, studies have indicated that successful completion rates and student perception of technical topics such as computer programming were deficient in online environments when compared to similar in class courses (Coffey, 2013; Courte, 2007). In addition, online completion rates appear to be deficient in online environments (Coffey, 2013).

Therefore, any evaluation tool applied to online education should consider adjusting for the following quality issues involved (Coffey, 2013).

Traditional teaching environments allow the instructor to take advantage of many methods to engage the student, involving both participation and feedback. Active learning is the concept of having the students participating in the instruction process, rather than passively listening to a lecture. In computer programming classes, active learning (such as programming exercises) has been effective in engaging students and increasing their retention (Ali, 2005). To accomplish this in an online course, instructors can take advantage of advanced web technologies and applications (Singh, Mangalaraj, & Taneja, 2010). Research has shown that online instruction can be as effective as a traditional classroom setting, given the appropriate use of technology – particularly when sufficient interactivity is employed (Durrington, Berryhill, & Swafford, 2006). Ideally, the interactions with the learning tools will also include feedback to the student. Ebrahimi (2011) evaluated the impact of early feedback on the success of students learning programming online. Early feedback on programming assignments consistently improved student performance.

One key area of difference between online teaching and in class teaching is that teachers and their students have less face-to-face interaction. Therefore, teachers have less impact on their student's learning approach in an online environment. The teacher can control the learning tools and components they provide, but students face a more flexible learning environment where they choose their learning strategies, learning pace, and direction. Depending on the student's orientation, an online environment can represent a complexity that challenges their skills in self-regulation and learning styles (Azevedo, Cromley, &Seibert, 2004; Corbeil, 2003; Guglielmino & Guglielmino, 2002; Kearsley, 2002; Yen & Liu, 2009; Zacharis, 2011). Students that are lacking in self-regulation are more subject to dropping their courses which results in higher drop-out rates when compared to in-class face-to-face environments (Allen & Seaman, 2005; Summers, Waigandt & Whittaker, 2005). Meyer (2003) found that students with independent and self-regulating styles tend to be better acclimated to online learning. One solution to supporting students with differing levels of independence and self-regulation is for educators to develop assignments, materials, and approaches that encourage the development of self-regulation and independent learning (Ambrose, et al. 2010, Weimer, 2002). Scaffolding is a term used to describe tools that educators develop and use to encourage and support independent and self-regulated behavior (Stachel, et al., 2013). Scaffolding is especially important in teaching computer programming to beginning learners. Beginning learners face many simultaneous challenges in their first programming course. They are expected to absorb a new world of interfacing with a language, an unfamiliar syntax, new form of logic and algorithmic thinking, and programming theory that can easily overwhelm and confuse a beginning learner, especially in an online environment, where face-to-face interaction is a challenge (Al-Imamy, Alizadeh, & Nour, 2006; Kelleher & Pausch, 2005; Stachel, et al., 2013).

The teaching of online programming courses is highly dependent upon the technology tools and software provided for the course (Fortune, et al., 2006). One area that is particularly challenging for online programming courses is providing hands-on lab opportunities with timely and personal feedback on programs developed by students (Jaggars, 2014; Kearsley, 2002; Steinbronn & Merideth, 2007). The availability of more sophisticated online tools has a direct implication on the choice of a programming language. Programming languages typically have tools that support using them. However, do they have tools to help with teaching them? For a

language to provide the best online class experience, there must be a rich set of tools to support the language and to assist in teaching the language.

PROGRAM SELECTION CRITERIA

A review of literature indicates that the following criteria be considered in a programming language selection quantitative model for online programming courses:

Must-Have Requirements

The following criteria were considered as necessary requirements before continuing to considered programming languages using the Quantitative Model to rank languages. These requirements tend to be a binary "yes or no" decision choice that eliminates languages from further consideration and ranking.

Platform

One practical consideration is that the chosen language must match the hardware and software platform requirements that the school and students will use. For example, if the school uses Apple hardware and OS/X, but the teaching tools for the language are only available for Linux or Windows, then the choice is untenable. However desirable the language might be, it won't be a practical choice if it doesn't match the school's existing or planned platform. The quantitative model doesn't aim to dictate any particular operating system or hardware. However, some products may only be available in certain configurations. Therefore, it is incumbent upon the school to have a preference and recommend it to the students.

Whatever the decision, it must be binding on the choice of programming language and associated software tools. Therefore, supporting the school's desired platforms will be a requirement for any candidate language.

Object Orientation

Ultimately, the object-oriented requirement is yes or no criteria determined by the faculty. The faculty can remove this restriction if they feel that the target market for their students does not require object-orientation.

Object-oriented programming is an important part of software development today. From 1993 to 2003, the use of the top procedural languages (C, Pascal, Basic) fell off considerably, while the popularity of object-oriented languages, like C++ and Java, rose dramatically (Chen, et al., 2005). It's important for students to understand object-orientation because even as object-oriented languages evolve, the core concepts of object-orientation remain (Zhu, 2012). Of the top five languages on the TIOBE index, only one is not object-oriented: the original C programming language. By the same token, JavaScript is a very popular and useful language for Web development applications and provides some object-oriented capabilities.

One of the questions at this point is whether object-orientation should be a requirement for the chosen programming language. The concepts introduced by object-oriented languages (such as polymorphism, inheritance, and encapsulation) are more challenging for students than the basic concepts like variables, operators, conditions, and looping (Milne & Rowe, 2002). This

creates a tension between the popularity of object-oriented languages and the challenges in teaching them. One potential bonus from taking the object-oriented approach is that students can move more comfortably from object-oriented to procedural programming than the other way around (Zhu, 2012).

Interactive Development Environment

To be productive in the use of many of today's programming languages and situations, the availability of an integrated development environment (IDE) is essential (Spinellis, 2006). Most IDEs have a common set of features: project organization, a code editor, multiple windows and navigation, plus the critical feature of interactive debugging (Fesq, 2002). In a workgroup setting, the IDE also controls updates to the source code repository. The availability of IDEs is even more important when teaching object-oriented programming to beginning learners and should typically be closely related to instructional and learner needs (Moons & Backer, 2013; McIver, 2002; Uysal, 2014). That makes IDEs particularly useful in a business programming environment. The use of IDEs has become so prevalent that in business development settings, IDEs are one of the most used application development tools (Binstock, 2007). IDEs have dual importance in the selection of a programming language. They are not only a critical tool for the individual programmer, but they are also a critical element in business settings. Because of this dual importance, the availability of an IDE for the programming language is an absolute requirement in programming language selection.

Want-To-Have Requirements

The following requirements are considered as desirable and are included as ranking factors in the Quantitative Model for language selection. These requirements can be scored from 1 to 10 and can be assigned different percentage weights to identify the final language choices.

Marketability

An important aspect to consider when selecting programming languages to teach is how well the language aligns with the current IS business market needs (Farooq, 2014; Vitkute-Adzgauskiene & Vidziunas, 2012). Chen, et al. (2005) provide an example of the importance of IS programs maintaining relevance by keeping up with industry needs. They provided the example of the period 1993 to 2003 in which Fortran use was trending upward in its use by students, while its use in industry was on the decline. For this quantitative model, we decided that the programming language chosen should be a language that students are likely to use in business. We refer to this criterion as marketability.

Therefore, the languages that are most widely in use today, and in demand in the business environment are favored criteria. We want to prepare our students to understand a language that they are most likely to encounter. In addition, the most widely used languages typically have the most support by textbooks and other learning tools.

One of the most-used measures for determining the current trends in programming languages is the TIOBE index (Vitkute-Adzgauskiene & Vidziunas, 2012). The TIOBE index doesn't claim to show the best programming language. It assembles data on the most popular

languages in current use. Monthly changes in the index indicates trends over time in the popularity of languages.

A literature search reveals that there are few scholarly articles related to the current marketability of programming languages at any given point in time. One of the reasons is fairly obvious: scholarly articles tend to have a lead time for publishing that makes their information less timely. Although the TIOBE index is a well-regarded and well-researched measure, other indicators are possibilities. Articles are available in the popular press that rank programming languages by the number of job postings for positions requiring that language. Faculty committees can combine several of these job board summaries for a more comprehensive view. Job market demand is an indicator of the current marketability for a programming language.

The Popularity of Programming Language (PYPL) index uses raw data from Google Trends to determine how often people search for tutorials on each programming language (http://pypl.github.io/PYPL.html). If students are learning a language today, it is likely because they are planning to use it in the future. As such, the PYPL index is a leading indicator of where language use could be going.

Combining the TIOBE index, job market indicators, and PYPL gives a balanced perspective on the past, present, and future popularity of current programming languages. Use of a summary of those rankings can establish the candidate languages for the analysis as well their relative popularity.

Interactive Tools

In recent years, there has been a lot of growth in the population of non-traditional learners. In particular, distance learning has removed a big barrier to higher-level education. For this quantitative model, the availability of teaching aids that would work in an online environment was a highly-desired feature (El-Bishouty, Ogata & Yano, 2007).

Courseware (the learning software used for the course) is the ingredient that can provide interactivity and feedback. Given the importance of those features, choosing the right courseware is essential. Ideally, the courseware is coordinated with the textbook to introduce and reinforce concepts consistently (Hsieh, 2011; Huan, Shehane, & Ali, 2011)

Video Support

Thomas Edison once said, “It is possible to teach every branch of human knowledge with the motion picture” (Reiser, 1987, p. 11). The role of video in online learning continues to grow significantly. Online video is now an accepted tool for traditional and distance learners (DeCesare, 2014). Without an instructor in a classroom, the videos fill the role of presenting the information both visually and verbally. A step-by-step video greatly simplifies teaching programming languages, in particular (Singh, Mangalaraj, & Taneja, 2010). Research indicates that visual teaching approaches led to improved comprehension of programming and debugging tasks (Baecker, DiGiano, & Marcus, 1997; Naharro-Berrocal, et al., 2002). Research indicates that student outcome measures improved when videos supplemented instructions in introductory computer programming courses (Shehane & Sherman, 2014).

Textbook

Students gain leverage from having a textbook that has companion software designed to match its content (Hsieh, 2011; Huan, Shehane, & Ali, 2011). If there were no such products, the alternative would entail matching unrelated books and software. Fortunately, there are learning products available from most major publishers. Due to the availability of coordinated textbooks and tools, only those type of textbooks will be considered for adoption in this quantitative model.

Our implementation of the quantitative model is to consider the number of recently published textbook editions for each language. The textbook choices are further reduced by eliminating books that do not provide companion software. Recall that the goal at this stage is to choose the language, not necessarily a specific textbook. Rather than identifying a specific textbook at this evaluation point, the language with the most textbook options could be considered to have the best textbook.

QUANTITATIVE MODEL

The overall approach proposed for the quantitative model is a simplified form of Kepner-Tregoe (KT) analysis that quantifies language requirements (Ozturk, Coburn, & Kitterman, 2003). K-T analysis divides language requirements into musts and wants. A candidate language is required to have the must-have-criterion satisfied to avoid elimination before applying scores. Want-to-have criteria determine the evaluation of the remaining choices. The best choice receives a score of 10, with other choices ranked proportionally to that top score.

Each want-to-have criterion is weighted and the total of criterion weights add up to 100 percent. Personal bias can affect the weightings as well as judgment regarding the rankings in each category. Reaching consensus on these numbers can be difficult, but it makes the resulting solution very clear. As a starting point, marketability can have half of the weight and language learning tools availability can be the other half of the weight.

If a small change in the weights would result in a different winner, then there should be a serious review to determine if the weightings are appropriate. Consider the case where one language is rising in use while another is declining. There is likely to be a crossover point where they are roughly equivalent in popularity. When this happens, it is important to consider the trend line. Preference should go to the language that is on the rise in terms of usage.

Table 1 compiles the results for each of the quantitative model want-to-have criterion. The weighted score is then computed and will be used to determine the best choice. The recommended weights displayed below are examples only.

The language with the highest weighted score is the best choice, given the accepted assumptions inherent in the model. The use of normalized scores, on a scale of 1 to 10, permit valid comparisons from year to year. Table 1 only shows three languages, but can easily include more candidates.

MUST-HAVE-CRITERIA

The university selected was making its newly designed Information Systems curriculum available online. To create the online version of the introductory business programming course, the initial task was to choose a programming language for the course. A faculty committee evaluated each language in terms of meeting the must-have-criteria. Based on the evaluation,

only those programming languages that met the following must-have-criteria were considered for scoring in the quantitative model.

Platform Support

At this point, the committee began culling the list by confirming the must-have features. The university selected supports students using both Windows and Mac computers. This is an issue with C#, Visual Basic, and Objective-C. C# is a Microsoft language designed to work with their operating system and .NET quantitative model. Support of C# and .NET use on other platforms is dependent on third-party software emulation (Binstock, 2004). Visual Basic has the same issues, being a Microsoft-specific implementation. Objective-C has subsisted on the fact that it is the language Apple chose for development on both OS/X and iOS. The faculty committee avoided selecting these languages due to their platform-specific issues.

Object-Oriented Support

Object-orientation was a firm requirement of the selected university, eliminating C from the list going forward. In addition, PHP is not fully object-oriented, but is a language that can support object creation; as such it was eliminated from further consideration. All other candidate languages provided adequate support of object-oriented programming and remained in consideration.

Availability of an IDE

There are multiple IDEs available for JavaScript, with Eclipse being the most popular. Eclipse is also available for C++ (eWeek, 2011). JetBrains, which also makes a popular Java IDE, develops the PyCharm IDE for Python (eWeek, 2010). Visual Basic uses the Visual Studio development environment produced by Microsoft (Heller, 2013). All remaining languages have at least one IDE available to work with them, which means that these languages can continue in the evaluation.

WANT-TO-HAVE-CRITERIA

A faculty committee evaluated the following want-to-have-criteria using the quantitative model.

Most Marketable

Candidates consisted of a list of the most marketable languages. The first measure used was the recommended TIOBE index https://www.tiobe.com/tiobe-index//. Although it is a monthly snapshot, the index also provides a historical graph which tells more than the rankings alone. A graph helps visualize and better understand trending.

Other marketing data is also available on the internet that can be used to assess marketability. The committee used Computerworld's top 10 list and IT CareerFinder.com's list of the most marketable programming languages. Finally, the study also used the current rankings from the PYPL index. Even though languages like HTML, XML, and SQL are popular, they

were eventually eliminated as candidates for an introductory programming course due to the committee's requirement that only object-oriented languages be considered. JavaScript is not widely considered as an object-oriented language, mainly due to lack of true inheritance. However, JavaScript was still considered in the analysis because it can support an object-oriented approach because it supports inheritance through prototyping as well as properties and methods.

Table 2 summarizes the data from the preceding sources and shows the results that formed the initial candidate list:

Best Interactive Tools

The university had been using a product from a major publisher for online instruction in computer programming. The product being used had exercises that were interactive and consisted of small programming challenges that require students to type the solutions into the software package. The software also gave the students instant feedback. The code was checked and either accepted as correct or the student received feedback about mistakes and possible corrections. The dual benefits of interactivity and rapid feedback met student needs. The faculty committee considered the capabilities of the existing product as essential and so other similar products were considered in the evaluation. The programming languages that were supported by the interactive tools were Java, C++, Python, and Visual Basic. The list of candidate languages focused on these four because of the interactive tools' availability.

5.2.3 Best Videos

Online informational videos are widely available, particularly from sites like YouTube (DeCesare, 2014). But it would be an ongoing chore to match assorted online videos with the content of the textbook and course. A better solution would be textbook supportive videos. Fortunately, textbooks were found that supported videos. These videos were available for a variety of books covering all programming languages remaining in the committee's analysis. Most of the textbooks contained notations that indicated the videos available for specific topics. The videos were typically short, topic-related videos of the kind shown to enhance learning, particularly for computer programming (Shehane & Sherman, 2013).

5.2.4 Best Textbook

Choosing the right textbook for an online computer programming course is essential (Huan, Shehane, & Ali, 2011). However, because the features of an interactive product tool were so highly desired, the faculty committee only considered books that had the companion interactive tool feature. The list of possible textbooks was limited to those that supported both features. The textbook titles varied by language as summarized in Table 3.

The faculty committee noted that the rankings of the languages by number of titles was also basically consistent with the marketability rankings.

It is worth mentioning that term length was a factor because the course in question runs for 9 weeks. Some books are suited for shorter courses, while others are suited for semester length.

5.8 Language Choice

At this point, all remaining languages considered had satisfied the must-have conditions. All languages rated equally for the best interactive tools and best videos. Clearly, there could be differences in the quality and number of videos and the completeness of the interactive exercises. However, the faculty committee deferred investigating these details with the proviso that they could investigate these areas later if necessary. For best textbook choice, each of the languages was being scored in proportion to the number of titles available. The resulting scores and weighted calculations are shown in Table 4.

From the popularity rankings to the number of textbook choices, Java is the top choice. With the other factors considered, it was a close finish between Java and C++. But on the TIOBE index, Java's usage percentage is higher than the other three languages combined.

It could have been a requirement that the programming language chosen for this course would have to be compatible with the choices for other courses in the curriculum. It would clearly be preferable to use the same language across many courses. The students would have a better chance at mastery by re-using, practicing, and adding to their knowledge in a series of courses. Coordination among courses can become problematic, if the analysis for a given course indicates that a different language would be superior to that used by the other courses. Should that new analysis precipitate a reconsideration of the language for the other courses? It could be that the common language is still the second-best choice for the course. Some situations might dictate that two different languages are the best solution. The quantitative model can't resolve that issue, but it can contribute by ranking the alternatives for each course.

CONCLUSIONS AND FUTURE RESEARCH

Java was the recommended programming language for the new online course at the selected university. A faculty committee reviewed and approved the quantitative model analysis. A faculty member developed the course using Java and faculty members are currently teaching the course online.

Because programming languages evolve and their usage shifts, the faculty committee needs to evaluates their choice of programming language for the online business programming course on an ongoing basis. In addition, the popularity of a language is only a partial consideration because as a language is emerging in popularity, there may be relatively few learning tools to support it. Even if it were incrementally more popular than another language, the lack of tools could keep it from the top rank overall. However, as the tool support grows, the scales could tip in favor of shifting to that language.

To provide structure and consistency to this process, the authors have proposed a quantitative model for evaluating the available languages on the market. This quantitative model focuses on choosing the language that is most relevant for business use and online delivery with the strongest set of teaching tools.

This article only considered the types of tools discovered through our current research. Just as computer languages will evolve, the tools to support online learning will also evolve and provide an increasingly rich interactive experience. Keeping abreast of developments in interactive learning will be as important as the language delivered with those tools.

REFERENCES

Ali, S. (2005). Effective teaching pedagogies for undergraduate computer science. *Mathematics and Computer Education*, 39*(3)*, 243-257.

Al-Imamy, S., Alizadeh, J. & Nour, M. (2006). On the Development of a Programming Teaching Tool: The effect of teaching by templates on the learning process. *Journal of Information Technology Education*, 5, 1-13.

Allen, I. E., & Seaman, J. (2005). *Growing by degrees—Online education in the United States.* Needham, MA: The Sloan Consortium.

Allen, I. E., & Seaman, J. (2010). *Class differences: Online education in the United States,2010.* Needham, MA: Babson Survey Research Group.

Ambrose, A. A., Bridges, M. W., DiPietro, M., Lovett, M. C., & Norman, M. K. (2010). *How learning works: Seven research-based principles for smart teaching.* San Francisco: Jossey-Bass.

Azevedo, R., Cromley, J. G., & Seibert, D. (2004). Does adaptive scaffolding facilitate students' ability to regulate their learning with hypermedia? *Contemporary Educational Psychology*, 29, 344–370.

Baecker, R., DiGiano, C., & Marcus, A. (1997). Software visualization for debugging. *Communications of the ACM*, 40(4), 44-54.

Binstock, A. (2004). Mono brings portability to .NET. *Software Development Times*, (107), 35.

Binstock, A. (2007). Java IDEs perk up. *InfoWorld*, 29(13), 21-26.

Brower, H. H. (2003). On emulating classroom discussion in a distance-delivered OBHR course: Creating an on-line community. *Academy of Management Learning and Education*, 2(1), 22-36.

Chen, Y., Dios, R., Mili, A., Wu, L., & Wang, K. (2005). An empirical study of programming language trends. *IEEE Software, 22*(3), 72-78. doi:http://dx.doi.org/10.1109/MS.2005.55

Clarke , S. (2001). Evaluating a new programming language. *In 13th Workshop of the Psychology of Programming Interest Group*, 275–289.

Coffey, J.W. (2013). Perspectives Regarding Computer Science Curriculum Delivery through Distance Education at Regional Universities. *The International Journal of Technology*, Knowledge, and Society, *8*(4), 73-82.

Corbeil, J. R. (2003). *Online technologies, self-efficacy, self-directed learning readiness, and locus of control of learners in a graduate-level web-based distance education program.* (Unpublished doctoral dissertation). University of Houston, Houston, TX.

Courte, J.E. 2007. Comparing Student Acceptance and Performance of Online Activities to Classroom Activities. *Proceedings of SIGITE'07*, pp. 185–189. October 18–20, 2007, Destin, Florida, USA.

DeCesare, J. A. (2014). The expanding role of online video in teaching, learning, and research. *Library Technology Reports*, 50(2).

Durrington, V. A., Berryhill, A., & Swafford, J. (2006). Strategies for enhancing student interactivity in an online environment. *College Teaching*, 54(1), 190-193.

Ebrahimi, A. (2011). How does early feedback in an online programming course change problem solving?. *Journal Of Educational Technology Systems*, *40*(4), 371-379.

Eclipse Turns 10 Tracing the Beginnings to a Little Java IDE 853952. (2011, November 2). *eWeek*.

El-Bishouty, M. M., Ogata, H., & Yano, Y. (2007). PERKAM: Personalized knowledge awareness map for computer supported ubiquitous learning. *Journal of Educational Technology and Society*, 10*(3)*. 122-134.

Ellis, R. D. & Kurniawan, S. H. (2000). Increasing the usability of online information for older users - a casestudy in participatory design. *Instructional Journal of Human-Computer Interaction*, 12*(2)*, 263–276.

Farooq, M., Khan, S., Farooq, A., Islam, S., & Abid, A. (2014). An evaluation quantitative model and comparative analysis of the widely used first programming languages. *Plos One*, 9*(2)*, 1-25. Retrived from http:://plospne.org.

Feldman, M.B. (1992) Ada experience in the undergraduate curriculum. *Communications of the ACM*. 35*(11)*, 53–67.

Fesq, B. J. (2002). Toward a consistent ide: Eclipse Platform/IDE. *New Architect*, 7*(9)*, 48-50.

Fortune, M.. et al. (2006) "A comparison of online (high tech)and traditional (high touch) learning in business and commun ication", *Journal of Education for Business*, 2, 210-214.

Gries, D. (1974) What should we teach in an introductory programming course?. In Proceedings of the 4th SIGCSE Technical Symposium on Computer Science Education. *ACM Press* :81–89.

Guglielmino, L. M., & Guglielmino, P. J. (2002). *Learner characteristics affecting success in electronic distance learning.* In H.B. Long & Associates, Twenty-First Century Advances in Self-Directed Leaning. Boynton Beach, FL: Motorola University Press.

Gupta, D. (2004) What is a good first programming language? *Crossroads*. 10*(4,* 7.

Heller, M. (2013). Review: Visual studio 2013 reaches beyond the IDE. *InfoWorld*.Com.

Hsieh, S. (2011). Effects of Cognitive Styles on an MSN Virtual Learning Companion System as an Adjunct to Classroom Instructions. *Educational Technology & Society*, 14*(2)*, 161–174.

Huan, X., Shehane, R., & Ali, A. (2011). Teaching computer science courses in distance learning. *Journal of Instructional Pedagogies*, 6, 1-14.

Jaggars, S. S. (2014). Choosing between online and face-to-face courses: Community college student voices. *American Journal of Distance Education*, 28*(1)*, 27-38.

Jenkins, T. (2001). *Themotivation of students of programming*.MSc Thesis submitted to the University of Kent.

JetBrains Strikes Python Developers with PyCharm 10 IDE 304127. (2010, October 14). *eWeek*.

Kearsley, G. (2002). Is online learning for everybody? *Educational Technology*, 42*(1)*, 41–44.

Kelleher, C., & Pausch, R. (2005). Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Computing Surveys* 37, 83–137.

Kolling, M. (2013). *This much I know: thoughts on the past, present and future of educational programming tools.* In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13), 5–6.

McIver, L. (2002) *Evaluating languages and environments for novice programmers.* In 14th Workshop of the Psychology of Programming Interest Group:100–110.

Meyer, K. (2003). The Web's impact on student learning. *T.H.E. Journal*, 30*(5)*, 14–24.

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming--views of students and tutors. *Education and Information Technologies*, 7(1), 55-66. doi:http://dx.doi.org/10.1023/A:1015362608943

Moons, J., & Backer, C. D. (2013). The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education*, 60, 368-384.

Mosca, J.B., Ball, D.R., Buzza, J.S., Dpaul, D.P. (2010). A comprehensive student-based analysis of hybrid courses: student preferences and design criteria for success. *Journal of Business & Economics Research*, 8*(5)*, 7-21.

Naharro-Berrocal, F., Pareja-Flores, C., Urquiza-Fuentes, J., & Velazquez-Iturbide, J. A. (2002). *Approaches to comprehension-preserving graphical reduction of pro-gram visualizations*. Paper presented at the Proceedings of the 2002 ACM symposium on applied computing.

Ozturk, O., Coburn, M.B., & Kitterman, S. (2003). *Conceptualization, Design, and Implementation of a Static Capacity Model*. Proceedings of the 35th Conference on Winter Simulation: Driving Innovation (pp. 1373-1376). New York, NY: ACM Press.

Parker, K.R., Ottaway TA, Chao JT, Chang J (2006) A Formal Language Selection Process for Introductory Programming Courses*, Journal of Information Technology Education*. 5, 133–151.

Schneider GM (1978). *The introductory programming course in computer science: ten principles*. In Papers of the 9th SIGCSE/CSA Technical Symposium on Computer Science Education. ACM Press, 107–114.

Reiser, R.A. (1987). "*Instructional technology: a history",* in Gagne, R.M. (Ed.), Instructional Technology: Foundations, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 11-48.

Shehane, R. & Sherman, S. (2014). Visual teaching model for introducing programming languages. *Journal of Instructional Pedagogy, 14*, 1-8.

Singh, A., Mangalaraj, G., & Taneja, A. (2010). Bolstering teaching through online tools. *Journal of Information Systems Education, 21(3)*, 299-311.

Spinellis, D. (2006). Choosing a programming language. *IEEE Software, 23(4)*, 62-63. doi:http://dx.doi.org/10.1109/MS.2006.97

Stachel, P., Marghitu, D., Brahim, T.B., Sims, R., Reynolds, L., & Czelusniak, V. (2013). Managing cognitive load in introductory programming courses: a cognitive aware scaffolding tool. *Journal of Integrated Design and Process Science,* 17*(1),* 37-54. DOI 10.3233/jid-2013-0004

Steinbronn, P.E. & Merideth, E.M. (2008) "Perceived Utility of Methods and Instructional Strategies Used in Online and Face-to-face Teaching Environments." *Innovative Higher Education*, 32, 265 -278.

Stroustrup, B. (2009) *Programming in an undergraduate CS curriculum*, In Proceedings of the 14th Western Canadian Conference on Computing Education (WCCCE '09). ACM, New York: 82–89.

Stroustrup, B. (2010) Viewpoint: What should we teach new software developers? Why? *Communications of the ACM*, 53*(1)*: 40–42.

Summers, J. J.,Waigandt, A. &Whittaker, T. A. (2005). A comparison of student achievement and satisfaction in an online versus a traditional face-to-face statistics class. *Innovative Higher Education*, 29(3), 233–250.

Uysal, M. P. (2014). Interviews With College Students: Evaluating Computer Programming Environments For Introductory Courses. *Journal of College Teaching & Learning, 11*(2), 59-70.

Vitkute-Adzgauskiene, D., & Vidziunas, A. (2012). Problems in choosing tools and methods for teaching programming. *Informatics in Education, 11*(2), n/a.

Weimer, M. (2002). *Learner-centered teaching: Five key changes to practice.* San Francisco; Jossey-Bass.

Wu, P. , & Chang, P. (2011). JavaScript for teaching accelerated programming basics courses. *Journal of Applied Global Research*, 4*(8)*, 27-38.

Yen, H. J., & Liu, S. (2009). Learner autonomy as a predictor of course success and final grades in community college online courses. Journal of Educational Computing Research, 41*(3)*, 347–367.

Zhu, X. (2012). Teaching adaptability of object-oriented programming language curriculum. *International Education Studies, 5(4)*, 237-242.

APPENDIX

Table 1. Model Ranking Results

	Most Marketable	Best Interactive Tools	Best Videos	Best Book	Score
Weight	*50%*	*30%*	*10%*	*10%*	*100%*
Language 1	10	10	8	10	9.8
Language 2	7	9	10	9	8.1
Language 3	3	5	7	8	4.5

Table 2. Most Marketable Results

TIOBE	Computerworld	ITCareerFinder	PYPL	Weighted Summary
C	Java	Java	Java	Java
Java	JavaScript	JavaScript	PHP	C++
Objective-C	C++	C++	C#	C#
C++	C#	C#	Python	JavaScript
C#	XML	XML	C++	C
PHP	Perl	C	C	Python
Visual Basic	Python	Perl	JavaScript	PHP
Python		Python	Objective-C	Objective-C
JavaScript			Ruby	Perl
			Visual Basic	Visual Basic

Table 3. Textbook Results

Language	Number of Titles
Java	8
C++	7
Python	3
Visual Basic	2

Table 4. Language Choice Results

	Most Marketable	Best Interactive Tools	Best Videos	Best Book	Score
Weight	*50%*	*30%*	*10%*	*10%*	*100%*
Java	10	10	10	10	10
C++	9	10	10	9	9.4
Python	5	10	10	5	7
Visual Basic	1	10	10	3	4.8